

```

1 public class Q3 {
2     public static void main(String[] args) throws OverflowException
3         , UnderflowException {
4         BoundedStack<Integer> stack = new BoundedStack<>(Integer.
5             class, 12);
6
7         // Push elements into the stack
8         int[] elements = { 2, 9, 3, 1, 8, 9, 0, 7, 8, 4, 5, 3 };
9         for (int element : elements) {
10             stack.push(element);
11         }
12
13         // Reverse the stack
14         BoundedStack<Integer> reversedStack = reverseStack(stack);
15
16         // Generate unique queue
17         Queue<Integer> uniqueQueue = generateUniqueQueue(
18             reversedStack);
19
20         // Other method calls...
21
22         // Time complexity analysis:
23         // - main method: O(n)
24     }
25
26     public static <T> BoundedStack<T> reverseStack(BoundedStack<T>
27         stack) throws OverflowException, UnderflowException {
28         Queue<T> queue = new Queue<>();
29
30         // Step 1: Pop elements from the stack and enqueue into the
31         queue
32         while (!stack.isEmpty()) {
33             T element = stack.pop(); // O(1)
34             queue.enqueue(element); // O(1)
35         }
36
37         BoundedStack<T> reversedStack = new BoundedStack<>(stack.
38             getType(), stack.getStack().length);
39
40         // Step 2: Dequeue elements from the queue and push into
41         the reversed stack
42         while (!queue.isEmpty()) {
43             T element = queue.dequeue(); // O(1)
44             reversedStack.push(element); // O(1)
45         }
46
47         return reversedStack;
48         // Overall time complexity of reverseStack method: O(n)
49     }
50
51     public static <T> Queue<T> generateUniqueQueue(BoundedStack<T>
52         stack)
53         throws OverflowException, UnderflowException {
54         Queue<T> QueueRepeated = new Queue<>();
55         Queue<T> uniqueQueue = new Queue<>();
56
57         while (!stack.isEmpty()) {

```

```

50     T value = stack.pop(); // O(1)
51     boolean flag = true;
52
53     while (!uniqueQueue.isEmpty()) {
54         if (value == uniqueQueue.peek()) {
55             flag = false;
56         }
57
58         Queuerepeated.enqueue(uniqueQueue.dequeue()); //O(1)
59     }
60
61     if (flag == true) {
62         Queuerepeated.enqueue(value); // O(1)
63     }
64
65     while (!Queuerepeated.isEmpty()) {
66         uniqueQueue.enqueue(Queuerepeated.dequeue()); //O(1)
67     }
68 }
69
70 return uniqueQueue;
71 // Overall time complexity of generateUniqueQueue method: O(n^2)
72 }
73 }

```

Listing 1: Q3 time complexity analysis

Therefore, the overall time complexity is $O(n^2)$

```

1 public void addToCart(Tuple tuple) throws OverflowException {
2     shoppingCart.push(tuple);
3 }
4
5 public void updateCart() throws OverflowException,
6     UnderflowException {
7     for (int i = 1; i < dict.size(); i++) { // Theta(n)
8         Tuple current = dict.get(i); // Theta(1)
9         Tuple prev = shoppingCart.peek(); // Theta(1)
10
11         // Calculate prices after discount
12         double prevPrice = (prev.getPrice() * (1 - prev.getDiscount() / 100.0));
13
14         // Theta(1)
15         double currPrice = (current.getPrice() * (1 - current.getDiscount() / 100.0)); // Theta(1)
16
17         // Compare prices and update cart accordingly
18         if (currPrice < prevPrice) { // Theta(1)
19             shoppingCart.pop(); // Theta(1)
20             shoppingCart.push(current); // Theta(1)
21             System.out.println((i + 1) + "th step: " + shoppingCart
22 + " as " + current.getPrice() + " X " + current.getDiscount()
23 + " percent = " + currPrice + " is less then " + prevPrice);
24         }
25         else if (currPrice > prevPrice) { // Theta(1)
26             System.out.println((i + 1) + "th step: " + shoppingCart
27 + " as " + current.getPrice() + " X " + current.getDiscount()
28 + " percent = " + currPrice + " is greater then " + prevPrice);
29         }
30     }
31 }

```

```

23     }
24
25
26     else if (currPrice == prevPrice && current.getDiscount() >
27 prev.getDiscount()) { // Theta (1)
28         shoppingCart.pop(); // Theta (1)
29         shoppingCart.push(current); // Theta (1)
30         System.out.println((i + 1) + "th step: " + shoppingCart
31 + " as " + current.getPrice() + " X " + current.getDiscount() +
32 + " percent = " + currPrice + " and " + current.getBrand() + "
33 has a discount");
34     }
35
36     else if (currPrice == prevPrice && current.getDiscount() <=
37 prev.getDiscount()) { //Theta(1)
38         System.out.println((i + 1) + "th step: " + shoppingCart
39 + " as " + current.getPrice() + " X " + current.getDiscount()
40 + " percent = " + currPrice + " and " + prev.getBrand() + " has
41 a bigger discount");
42     }
43
44 }
45
46 @Override
47 public String toString() {
48     return shoppingCart.toString();
49 }
50
51 public static void main(String[] args) {
52     Q4 shopping = new Q4(); // Theta (1)
53
54     try {
55         shopping.addToCart(shopping.dict.get(0)); // Theta (1)
56         System.out.println("Shoes info: " + shopping.dict); // Theta
57 (1)
58         System.out.println("1st step: " + shopping.shoppingCart); //
59 Theta (1)
60
61         shopping.updateCart(); // Theta (n)
62     } catch (OverflowException | UnderflowException e) {
63         e.printStackTrace();
64     }
65 }

```

Listing 2: Q4 time complexity analysis

Therefore, the overall time complexity is $\Theta(n)$